NPS52-84-011

NAVAL POSTGRADUATE SCHOOL

Monterey, California

A DECOMPOSABLE ALGORITHM FOR CONTOUR SURFACE DISPLAY GENERATION

Michael J. Zyda

August 1984

Approved for public release; distribution unlimited

Prepared for:

Chief of Naval Research
Arlington, VA 22217

NAVAL POSTGRADUATE SCHOOL
Monterey, California

Commodore R. H. Shumaker
Superintendent

D. A. Schrady
Provost

The work reported herein was supported in part by the Foundation Research Program of the Naval Postgraduate School with funds provided by the Chief of Naval Research.

This report was prepared by:

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER NPS52-84-011	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) A Decomposable Algorithm for Contour Surface Display Generation		5. TYPE OF REPORT & PERIOD COVERED 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Michael J. Zyda		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, CA 93943		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 61152N; RR000-01-10 N0001484WR41001
11. CONTROLLING OFFICE NAME AND ADDRESS Chief of Naval Research Arlington, VA 22217		12. REPORT DATE August 1984 13. NUMBER OF PAGES 29
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report)		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number)		

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This paper is a study of a highly decomposable algorithm useful for the parallel generation of a contour surface display. The core component of this algorithm is a two-dimensional contouring algorithm that operates on a single, 2 x 2 subgrid of a larger grid. A model for the operations used to generate the contour lines for a single subgrid is developed. The inadequacies of the currently published algorithms, with respect to contour line generation for a subgrid, are pointed out in a brief review of the available literature. A data structure,

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

S/N 0102-LF-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

the contouring tree, is introduced as the basis of a new algorithm for generating the contour lines for the subgrid. The construction of the contouring tree, and the completeness of an algorithm based upon the contouring tree, within the constraints of the contouring model are shown.

A Decomposable Algorithm for Contour Surface Display Generation ‡

Michael J. Zyda

Naval Postgraduate School,
Code 52, Dept. of Computer Science,
Monterey, California 93943

ABSTRACT

This paper is a study of a highly decomposable algorithm useful for the parallel generation of a contour surface display. The core component of this algorithm is a two-dimensional contouring algorithm that operates on a single, 2 x 2 subgrid of a larger grid. A model for the operations used to generate the contour lines for a single subgrid is developed. The inadequacies of the currently published algorithms, with respect to contour line generation for a subgrid, are pointed out in a brief review of the available literature. A data structure, the contouring tree, is introduced as the basis of a new algorithm for generating the contour lines for the subgrid. The construction of the contouring tree, and the completeness of an algorithm based upon the contouring tree, within the constraints of the contouring model, are shown.

Categories and Subject Descriptors: I.3.3 [**Picture/Image Generation**]: surface visualization; I.3.5 [**Computational Geometry and Object Modeling**]: data structures, planar contours, surface approximation, surface generation, surface representation, surfaces; I.3.7 [**Three-Dimensional Graphics and Realism**]: line drawings, line generation algorithms, surface plotting, surface visualization, surfaces;

General Terms: contouring, contouring tree, contour surface display generation;

1. Introduction

Contour surface display generation is one of the most frequently used graphics algorithms [Zyda,1984], [Zyda,1983], [Zyda,1982], [Zyda,1981], [Barry,1979], [Faber,1979], and [Wright,1979]. A contour surface is a visual display that represents all points in a particular region of three-space <x,y,z> which satisfy the relation f(<x,y,z>)=k, where k is a constant known as the contour level. The function f represents a physical quantity which is defined over the three-dimensional volume of interest. The visual display created by this algorithm is the collection of lines that belong to the intersection of both the set of points that satisfy the relation f(<x,y,z>)=k, and a set of regularly spaced parallel planes that pass through the region of three-space for which the

‡ This work has been supported by the NPS Foundation Research Program.

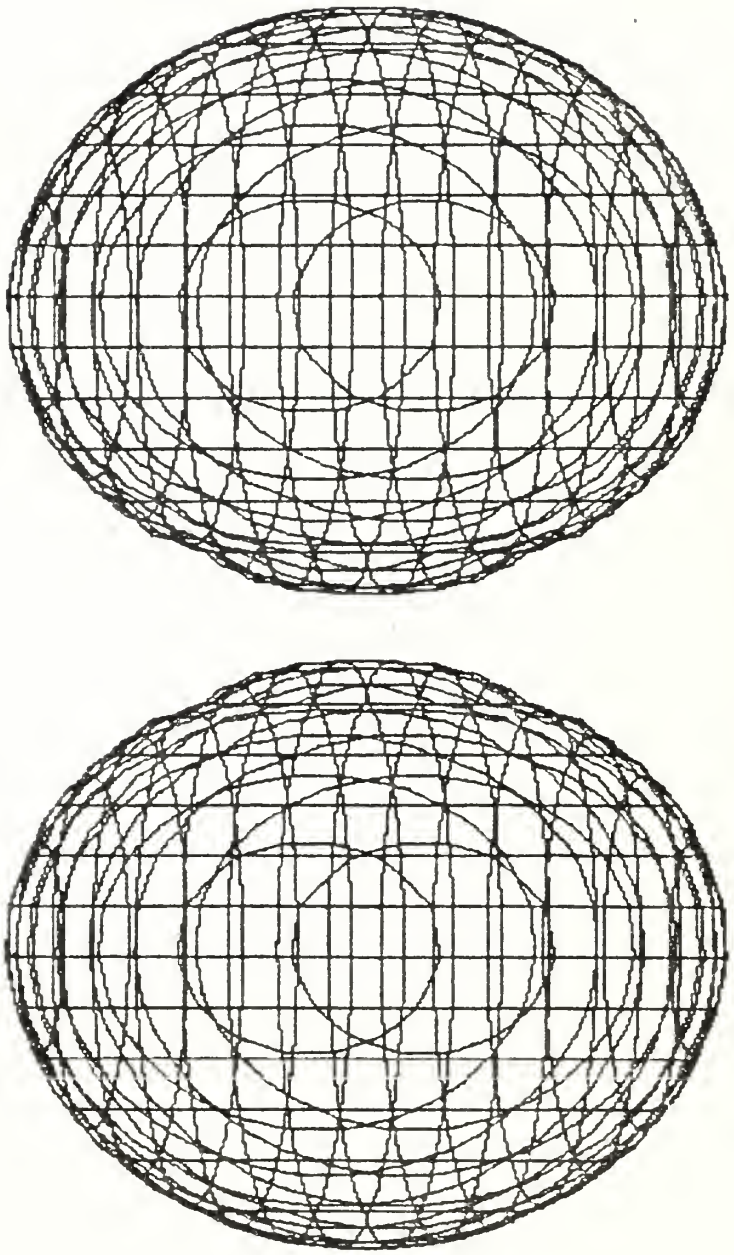


Figure 1
Contour Surface Display Generated from a Hydrogen Atom
Wavefunction Squared (3dxy orbital)

relation is defined.

For this study, the function f is approximated by a discrete, three-dimensional grid created by sampling that function over the volume of interest. The three-dimensional grid contains a value at each of its defined points that corresponds to the physical quantity obtained from the function, i.e. the value associated with point (x_0,y_0,z_0) is v_0, where $f(x_0,y_0,z_0)=v_0$. In order to minimize confusion, we will specify the value at a particular grid point (x,y,z) by a(x,y,z), and will specify the value at a particular point (x,y,z) of the function by f(x,y,z).

The visual display of the contour surface is created from this three-dimensional grid by taking two-dimensional slices of the grid, and constructing the two-dimensional, planar contours for each slice at the designated contour level. A slice of a three-dimensional grid is a planar, orthogonal, two-dimensional grid assigned a constant coordinate in three-space, i.e. an x-y slice of a(<x,y,z>) corresponds notationally to a(<x,y>) for a particular z coordinate. The two-dimensional, planar contours created are the lines that satisfy the relation a(<x,y,z>)=k for a particular planar coordinate, either x, y, or z, where again k is the constant contour level. If we contour all x-y slices of the three-dimensional grid at contour level k, we will have a stack of parallel contours approximating the contour surface, each planar set of contours corresponding to a particular z coordinate. If we contour all x-z slices of the three dimensional grid, we again will have a stack of parallel contours approximating the contour surface, each planar set of contours corresponding to a particular y coordinate. Likewise, if we contour all y-z slices of the three-dimensional grid, we will have a stack of parallel contours approximating the contour surface, each planar set of contours corresponding to a particular x coordinate. The assemblage of the three sets of parallel, planar contours, i.e. the simultaneous display of all the contours created for the x-y, x-z, and y-z planes of the three-dimensional grid, produces a "chicken-wire-like" contour surface display (see Figure 1). The three-dimensional contour surface display described in this study is created by such a procedure.

Given that the core of the contour surface display generation algorithm is the two-dimensional slice of the three-dimensional grid, it is best that we start our study with an understanding of the operations performed on that slice. Figure 2 shows a single, x-y, two-dimensional grid, with the contours drawn corresponding to contour level 50. Figure 3 shows that same two-dimensional grid, with the contours drawn corresponding to contour level 100. The two-dimensional grid of those figures is a 4 x 5 grid; it has four values in the x direction, and five values in the y direction. The goal of the two-dimensional contouring operation for such a grid is the determination of where lines are drawn on that grid given a fixed contour level k. In order to develop an intuitive feel for that determination mechanism, we restrict our focus to a small portion of the complete two-dimensional grid, the 2 x 2 subgrid. The 2 x 2 subgrid is defined to be that portion of the two-dimensional grid bounded by four adjacent grid points. In the two-dimensional grid of Figures 2 and 3, the lower, lefthand 2 x 2 subgrid is bounded by points (1,1), (2,1), (2,2), and (1,2). The upper righthand 2 x 2 subgrid of the same example is bounded by points (3,4), (4,4), (4,5), and (3,5).

2. A Model for Contouring the 2 x 2 Subgrid

The procedure used to generate the contours for a single 2 x 2 subgrid is the core part of two-dimensional contouring. If we compute the contours corresponding to contour level k for all 2 x 2 subgrids of a two-dimensional grid, then we will have determined the complete set of contours for that grid. Note

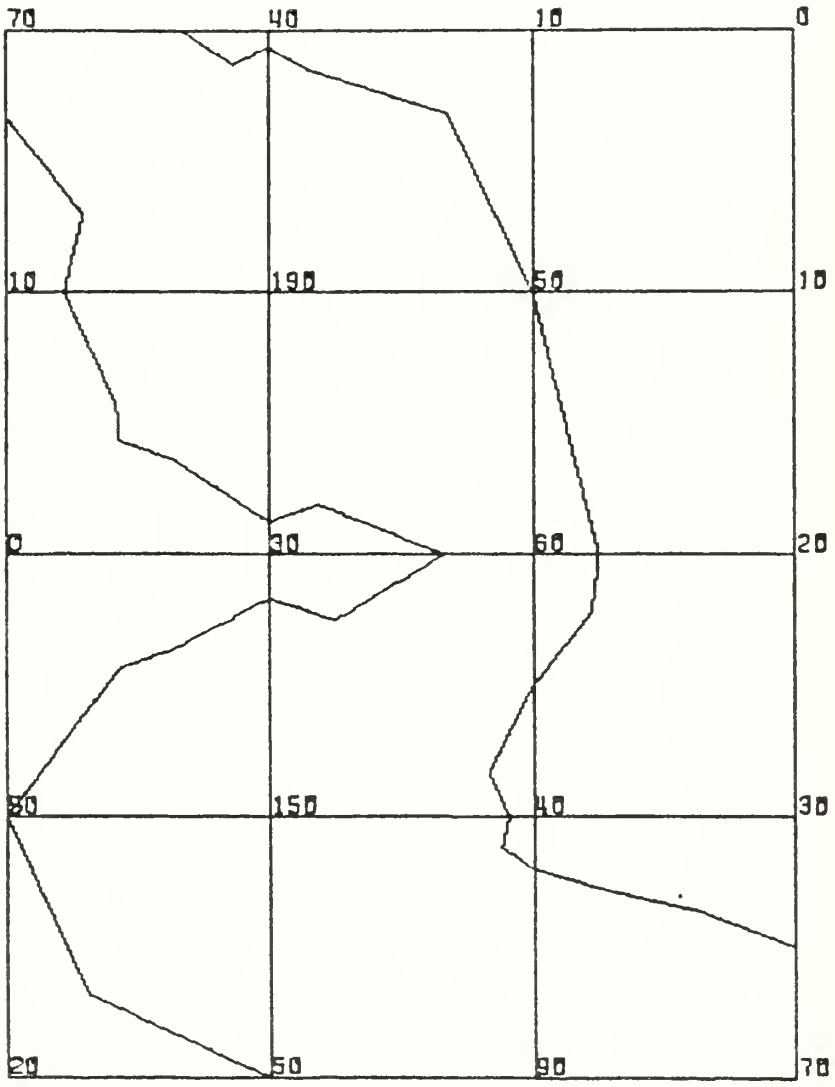


Figure 2
Example Contour Grid with Contours Drawn for Level 50

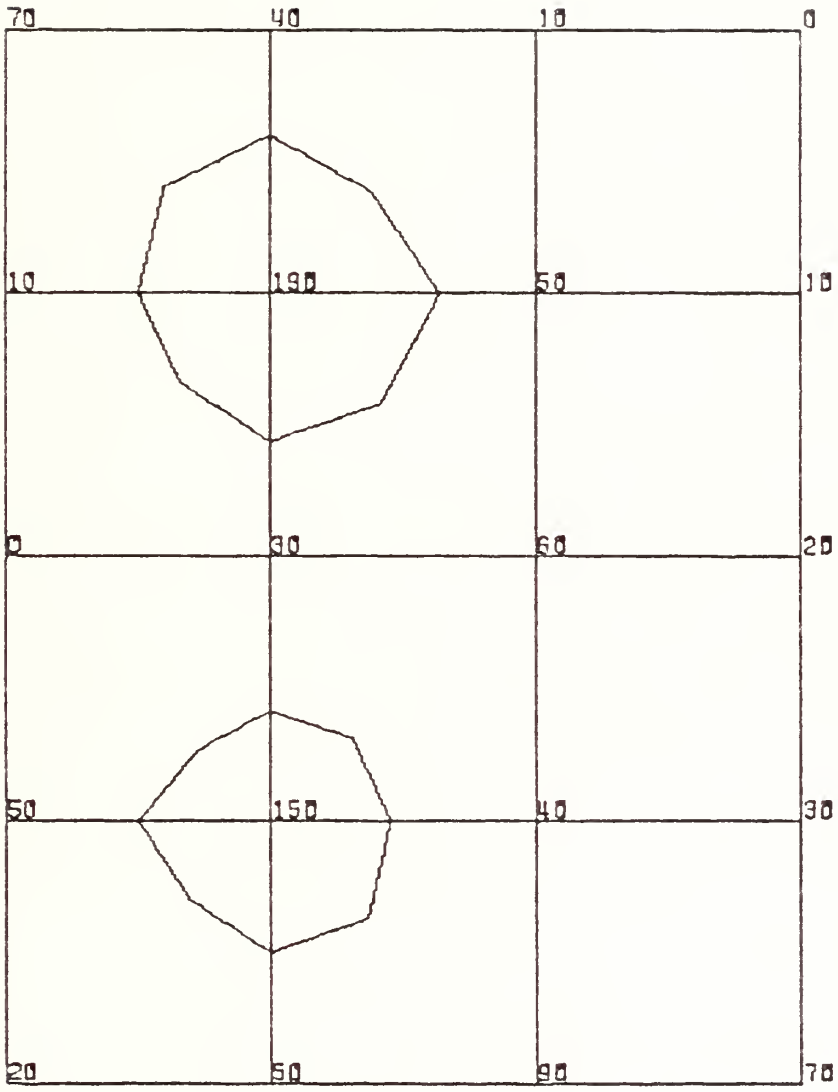


Figure 3
Example Contour Grid with Contours Drawn for Level 100

that this does not make any statement as to the efficiency of that picture, i.e. there can be duplicate copies of contours, particularly for contours drawn along the border of a 2 x 2 subgrid (shared edge). In order to provide an intuitive feel for contour generation on the 2 x 2 subgrid, we briefly summarize the operations that comprise that procedure in order to highlight potential problems.

The procedure used to generate the contours for a particular 2 x 2 subgrid first determines if any contours should be generated for that subgrid. That determination is based upon whether any of the subgrid's edges contain the desired contour level k. An edge contains contour level k if the value of that contour level is within the range of values defined by the grid points that comprise the edge.

The next part of the contour generation procedure for the 2 x 2 subgrid is the computation of the contour edge intersections for any subgrid edges shown to contain the contour level. The point of intersection is computed through linear interpolation, using the grid values assigned to the endpoints of the edge and their corresponding coordinates. The point of intersection represents the location on the subgrid edge corresponding to the contour level k.

The determination of the connectivity necessary to form the appropriate contours from the list of edge intersections is the next part of the contour generation procedure. Before attempting to describe the procedure that assigns those connectivities, we first examine the subgrid's contour crossing possibilities. We accomplish that by looking at Figure 4, which shows all possible ways for contours to cross or intersect a 2 x 2 subgrid.

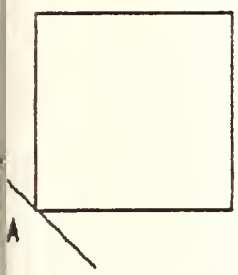
In Figure 4, there are ten cases, each of which belongs to one of three contour crossing categories: (1) single edge crossings of the 2 x 2, (2) double edge crossings of the 2 x 2, and (3) constant edge borders at the contour level for the 2 x 2. The ten cases are drawn according to the following small set of rules for contour crossings.

> (1) Contours are directed by the values associated with the edges, and are directed towards edge intersections.
>
> (2) For non-equivalued edges, if contours are indicated for a particular 2 x 2 subgrid, i.e. there are edges in the subgrid that contain the contour level, there is only one point of intersection for each edge of that subgrid.
>
> (3) Contours are continuous, i.e. if a contour enters a 2 x 2 subgrid, it must also leave that 2 x 2 subgrid.
>
> (4) Equivalued subgrid edges at the contour level are special cases, and are drawn in their entirety. The only exception to this rule is that constant valued 2 x 2 subgrids are not drawn. This is by convention.

The first rule, that of contours being directed by the values associated with the edges, and contours being directed towards edge intersections, means that one determines the placement of contours, and hence, the connectivity of the edge intersections, by using both the values assigned to the endpoints of each edge of the subgrid, and the computed intersections of that subgrid. The importance of this rule is twofold. First, it indicates that no outside forces or parameters direct contour placement. Second, the rule indicates that computed intersections are not the sole basis for determining the connectivity of the contours.

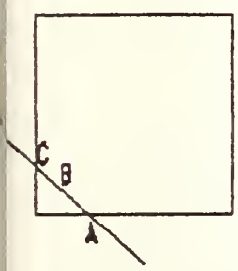
SINGLE CONTOUR CROSSINGS

CASE 1: CONTOUR TANGENT
TO THE 2 X 2



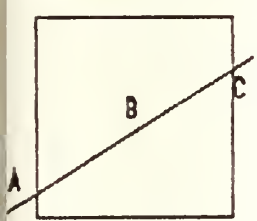
EXPECTED PICTURE:
DRAWPOINT A

CASE 2: ONE CONTOUR
THROUGH ADJACENT EDGES



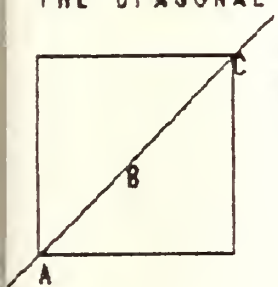
EXPECTED PICTURE:
SETPOINT A
DRAWTO B
DRAWTO C

CASE 3: ONE CONTOUR
THROUGH PARALLEL EDGES



EXPECTED PICTURE:
SETPOINT A
DRAWTO B
DRAWTO C

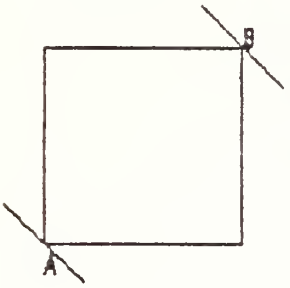
CASE 4: CONTOUR ACROSS
THE DIAGONAL



EXPECTED PICTURE:
SETPOINT A
DRAWTO B
DRAWTO C

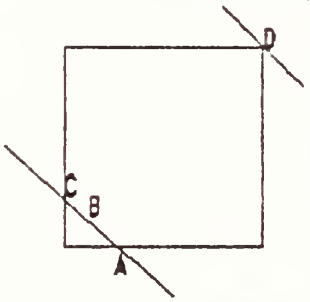
DOUBLE CONTOUR CROSSINGS

CASE 5: TWO CONTOURS
TANGENT TO THE 2 X 2



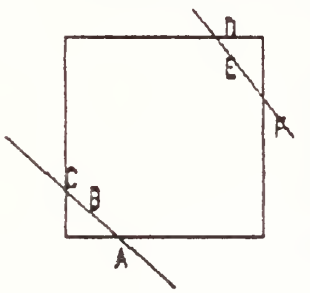
EXPECTED PICTURE:
DRAWPOINT A
DRAWPOINT B

CASE 6: ONE CONTOUR TANGENT,
ONE CONTOUR THROUGH ADJACENT EDGES



EXPECTED PICTURE:
SETPOINT A
DRAWTO B
DRAWTO C
DRAWPOINT D

CASE 7: TWO CONTOURS
THROUGH ADJACENT EDGES

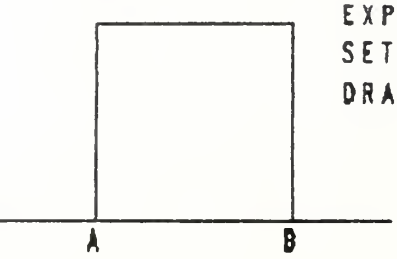


EXPECTED PICTURE:
SETPOINT A
DRAWTO B
DRAWTO C
SETPOINT D
DRAWTO E
DRAWTO F

FIGURE 4
ALL POSSIBLE CONTOUR CROSSINGS OF A 2 X 2 SUBGRID

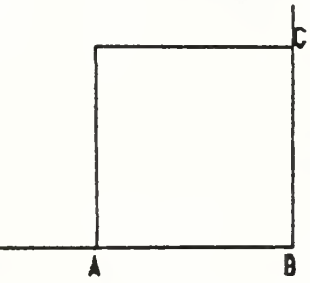
EQUIVALUED EDGES AT THE CONTOUR LEVEL

CASE 8: CONTOUR ALONG
ONE EDGE



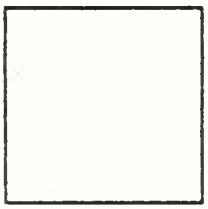
EXPECTED PICTURE:
SETPOINT A
DRAWTO B

CASE 9: CONTOUR ALONG
TWO EDGES



EXPECTED PICTURE:
SETPOINT A
DRAWTO B
SETPOINT B
DRAWTO C

CASE 10: CONSTANT 2 X 2



EXPECTED PICTURE:
NONE.

FIGURE 4 (CONTINUED)
ALL POSSIBLE CONTOUR CROSSINGS OF A 2 X 2 SUBGRID

The second rule, that of there being only one point of intersection for each edge of the 2 x 2 subgrid, means that for an edge intersected by a contour, there are no other points along that edge with contour value k. Since the range of values assigned to that edge is continuous and monotonic, the origin of this rule is clear. Note that this rule does not apply to equivalued edges.

The third rule, that of contours being continuous, means that if a contour enters a 2 x 2 subgrid, it must also leave that subgrid, i.e. the contour does not terminate inexplicably in the middle of the 2 x 2. A corollary of this rule, in combination with the first and second rules, states that contours entering a subgrid through an edge must leave via a different edge. Again, note that this corollary does not apply to cases with equivalued edges. This corollary holds though for contours tangent to grid points of the subgrid if we arbitrarily assign one of the subgrid edges to that grid point as the entering edge, and assign the other edge as the leaving edge.

The last rule, that of equivalued subgrid edges at the contour level being special cases and being drawn in their entirety, is a rule based on visual expectations for the contour for such a subgrid edge. The only exception to this rule, that of not drawing constant valued 2 x 2 subgrids, is adopted by general convention.

Once we have an idea of the types of contour crossings possible for a 2 x 2 subgrid, and once we have an outline of the rules used in composing those possibilities, we can then address the problem of forming a procedure for assigning connectivities to the computed edge intersections. Starting with the simplest cases of Figure 4, the equivalued edge cases, we clearly see that the connectivity generation procedure for subgrids containing such edges at the contour level is relatively simple once those equivalued edges have been detected. If we find that we have a "constant 2 x 2", we do not need to issue any coordinates or connectivities because by convention we have decided not to draw that case. The other two possibilities, the "contour along one edge", or the "contours along two edges" cases, are equally as simple. The only operation necessary once such cases have been detected is to issue coordinates and connectivities corresponding to the detected edges.

At first glance, given the edge intersections for a 2 x 2 subgrid, the connectivity generation procedure for the single contour cases of Figure 4 seems quite easy. It appears as if the only operation that has to be done is to issue coordinates and connectivities corresponding to the straight line between the two points of edge intersection. Such a procedure works well if we know that we have a single contour crossing the subgrid. The only single contour crossing case for which this will not work is the "contour tangent to the 2 x 2" case, which is an even simpler case for connectivity generation.

It is not until we consider the two contours crossing the subgrid cases of Figure 4 that we realize the potential for problems with the above single contour crossing procedure. A procedure based only on connecting edge intersections cannot differentiate between cases such as the "two contours tangent to the 2 x 2", and the "contour across the diagonal" cases. There are other similar connectivity generation problems evident for the two contours crossing cases. The "two contours through adjacent edges" case has four edge intersections. For that case, information needs to be provided to the connectivity generation procedure that determines which of three possible intersection pairs should be connected.

Now that we have established a background for the connectivity problem for contour crossings of the 2 x 2 subgrid, we can detail the procedure used to solve that problem. Before we describe that algorithm, we first briefly review

some of the problems cited in the literature for two-dimensional contouring.

3. Two-Dimensional Contouring Literature Problems

The literature on two-dimensional contouring, and the use of two-dimensional contouring for creating a contour surface display is extensive, encompassing a number of fields ([Zyda,1984], [Zyda,1983], [Zyda,1982], [Zyda,1981], [Dutton,1977], [Gold,1977], [Wright,1979], [Cottafava,1969], [Dayhoff,1963], [Faber,1979], [McLain,1974], [Sabin,1980], [Sutcliffe,1980], [Sutcliffe,1976a], and [Sutcliffe,1976b]. A thorough review of the historical development of two-dimensional contouring algorithms and the properties of those algorithms is found in [Sutcliffe,1980]. Many of the contouring algorithms presented in that study are flawed either in that they generate an incorrect picture for some contour crossing cases, or in that they require special handling for "problem" 2 x 2 subgrids. Some of the typical algorithm problems detailed are identical to those described above, i.e. they concern "degenerate points", grid points that are equal to the contour level, or "saddle points", 2 x 2 subgrids where there are ambiguities as to which points to connect. In all of the algorithms reviewed, no attempt is made to fit the special cases inside of a general algorithmic framework. This is quite evident for the subgrid having a saddle point. That contour crossing case is handled by selecting the two lines "for which the direction changes the least" when compared against neighboring subgrids [Sutcliffe,1980]. Again, this requires special algorithmic resolution. None of the papers attempts to build a general framework useful for the generation of the coordinates and drawing instructions for any 2 x 2 subgrid. The following section describes both a data structure, the contouring tree, and an algorithm for using that data structure, that provide both a coherent framework for 2 x 2 subgrid contouring, and a comprehensive resolution to the 2 x 2 subgrid crossing problem.

4. The Contouring Tree

A contouring tree is a data structure that represents the edge value relationships of a 2 x 2 subgrid in a form that permits the rapid generation of the contour display for any contour level contained within the represented subgrid (see Figure 5). The formulation of the contouring tree is based upon the observation that for any two-dimensional grid a continuous series of contour displays can be created for contour levels in the range of the minimum and maximum grid values (see Figure 6, and [Zyda,1984], [Zyda,1983], [Zyda,1982], and [Zyda,1981]).

The use of the contouring tree is outlined best with an example of a small two-dimensional grid. Figures 2 and 3 depict the contours generated for contour levels 50 and 100. The contours at level 100 are closed contours, forming simple, connected loops. The contours at level 50 are open contours. Figures 5 and 7 present the contouring trees created for two 2 x 2 subgrids of the 4 x 5 plane. The edges of the contouring trees correspond to the directed, downhill edges inscribed on the 2 x 2 subgrids of the figures. There are eight directed edges on each subgrid, four for the boundary edges and four for the edges to the subgrid's center point. The value used for the center point is the average of the four values comprising the corners of the 2 x 2 subgrid. (A reference as to the usefulness of the center point average value in generating smooth contours is found in [Sutcliffe,1980].) The edges of the contouring trees are ordered, maintaining the same counterclockwise ordering as in the original subgrids. A "1" under a node indicates that a setpoint display command should be generated for any coordinate that is created along an edge that has that connectivity on its lower valued node. A "0" indicates a drawto display command in a similar

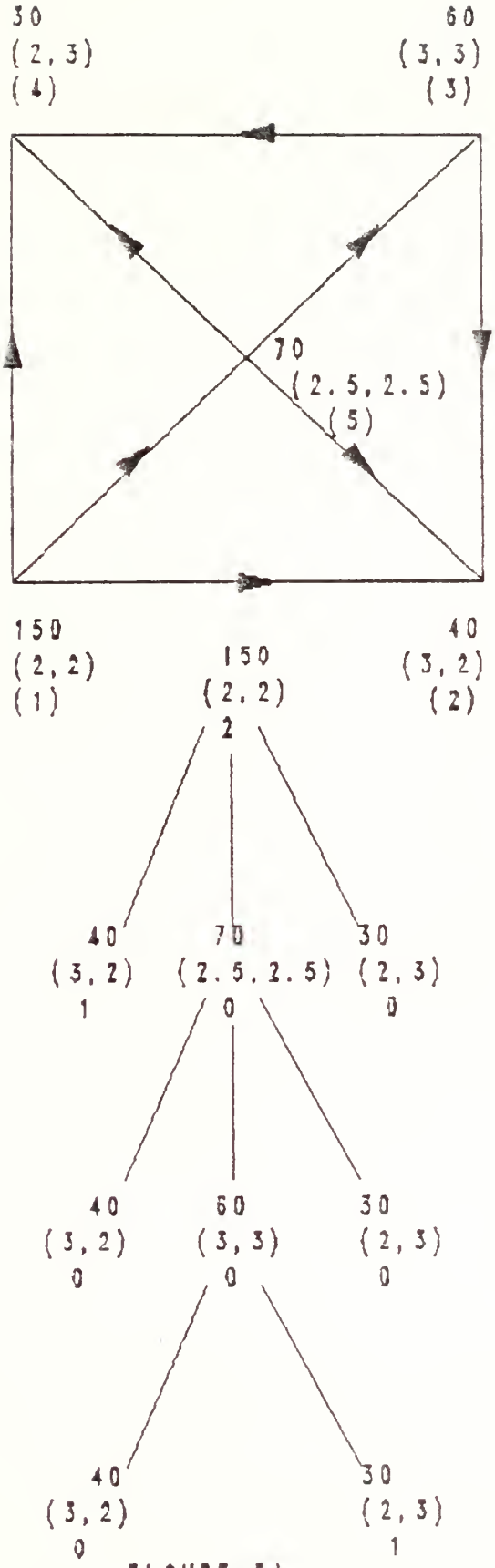


FIGURE 5A

SAMPLE CONTOURING TREE FOR A 2 X 2 SUBGRID

Level 50

X	Y	Z	D
2.9091	2.0000	1.0000	1
2.8333	2.1667	1.0000	0
3.0000	2.5000	1.0000	0
2.6667	3.0000	1.0000	1
2.2500	2.7500	1.0000	0
2.0000	2.8333	1.0000	0

Level 100

X	Y	Z	D
2.4545	2.0000	1.0000	1
2.3125	2.3125	1.0000	0
2.0000	2.4167	1.0000	0

Column D is the drawing command, ie. 1 = SETPOINT, 0 = DRAWTO.

Figure 5b
Coordinates Generated for Sample 2 x 2 Subgrid

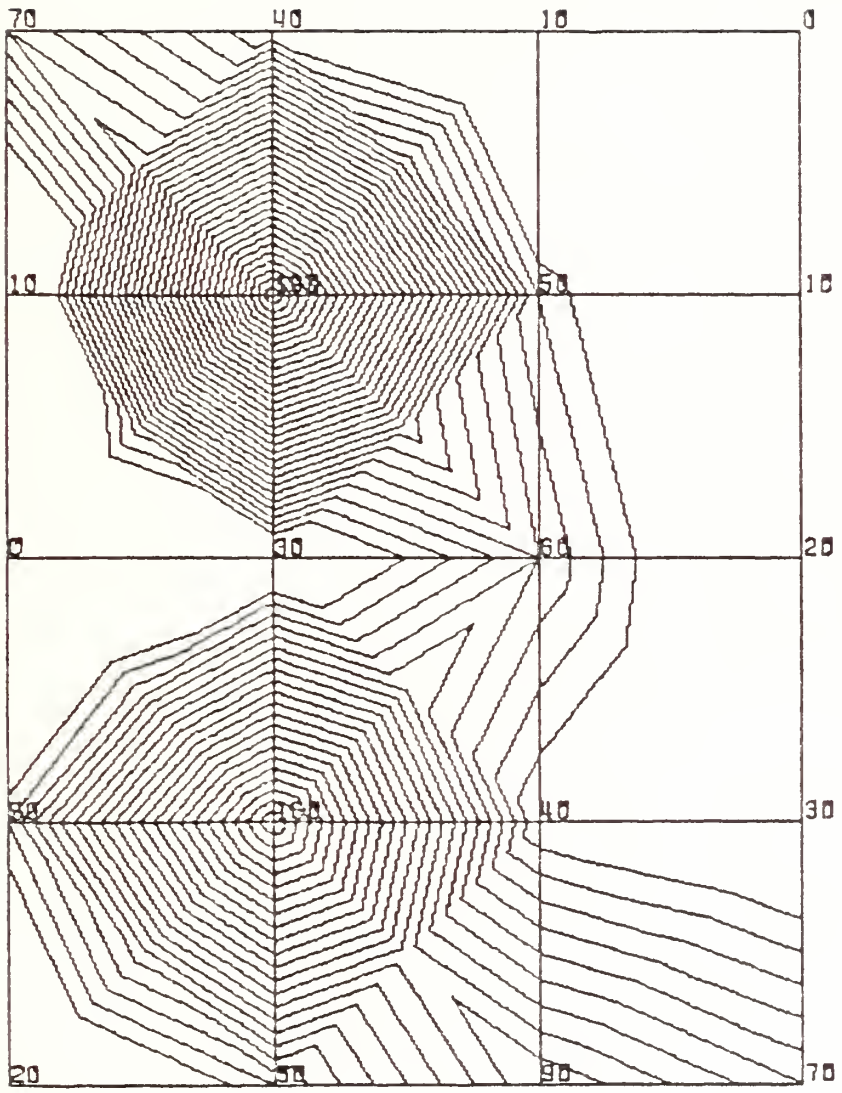
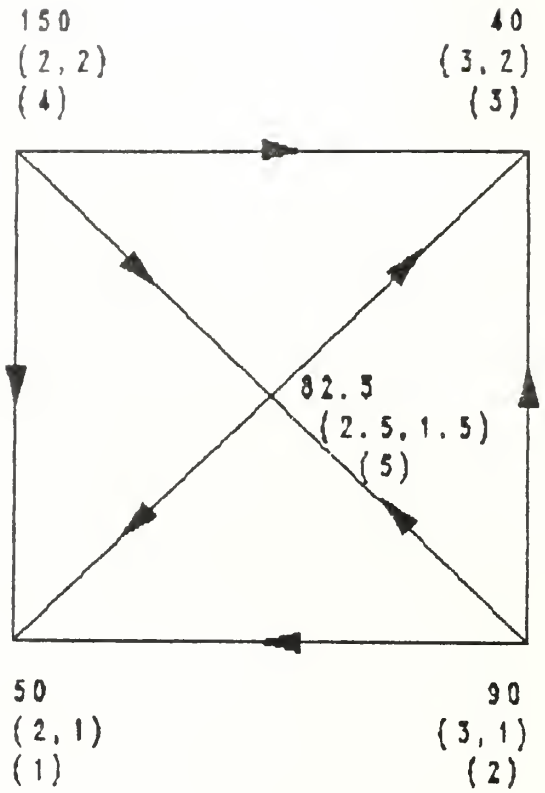


Figure 6
Example Contour Grid with Contours Drawn for Multiple Contour Levels



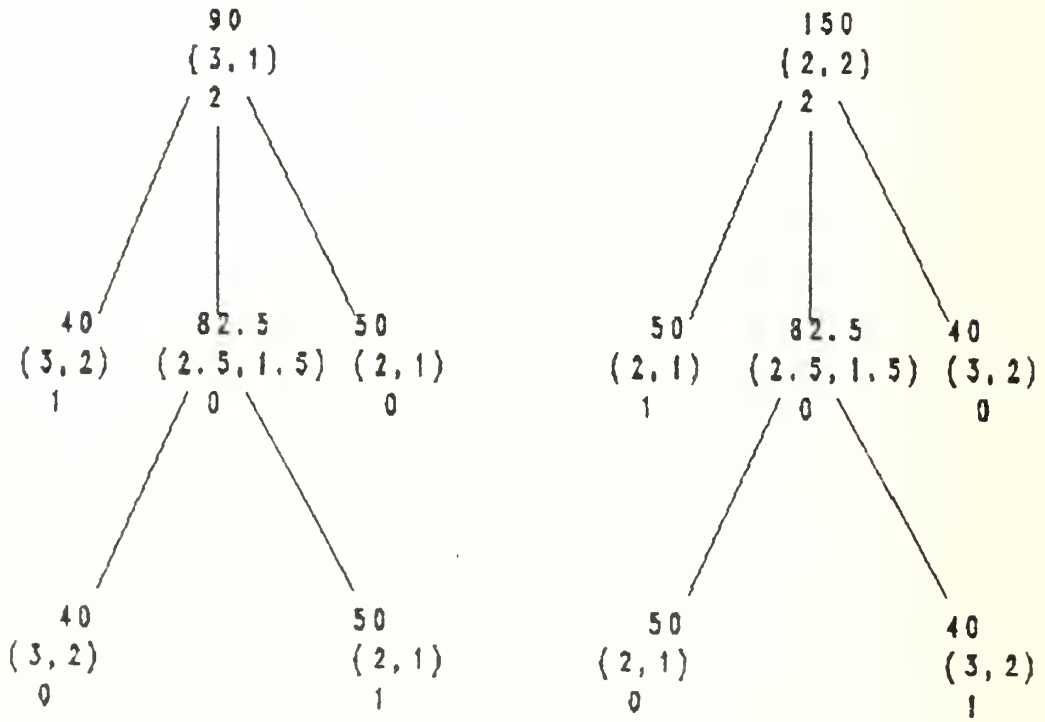


FIGURE 7A
SAMPLE CONTOURING TREE FOR A 2 X 2 SUBGRID WITH SADDLE POI

Tree rooted at value 90

Level 50

X	Y	Z	D
3.0000	1.8000	1.0000	1
2.8824	1.8824	1.0000	0
2.0000	1.0000	1.0000	1
2.0000	1.0000	1.0000	0

Level 100

X	Y	Z	D
no coordinates generated			

Tree rooted at value 150

Level 50

X	Y	Z	D
2.0000	1.0000	1.0000	1
2.0000	1.0000	1.0000	0
2.8824	1.8824	1.0000	1
2.9091	2.0000	1.0000	0

Level 100

X	Y	Z	D
2.0000	1.5000	1.0000	1
2.3704	1.6296	1.0000	0
2.4545	2.0000	1.0000	0

Column D is the drawing command, ie. 1 = SETPOINT, 0 = DRAWTO.

Figure 7b
Coordinates Generated for Sample 2 x 2 Subgrid with Saddle Point

fashion and a "2" indicates a drawpoint.

Display generation from a contouring tree is accomplished by performing a pre-order traversal of that contouring tree, producing a coordinate and drawing instruction whenever the desired contour level is found to be within the range of an edge of the contouring tree. A pre-order traversal visits the root, the left subtree, the middle subtree, and then the right subtree. An edge's range is defined to be the set of values between those associated with the nodes on either end of the edge. More precisely, we say a contour level is within an edge if the following condition holds:

lower_node's_value $\leq$ contour_level < higher_node's_value

For example, in Figure 5a at contour level 100, we issue coordinates and drawing instructions for the edges (2,2)-(3,2), (2,2)-(2.5,2.5), and (2,2)-(2,3). The drawing instruction issued for each of these edges is again the one associated with the lower valued node of the edge. The coordinate for each of these edges is generated by a linear interpolation of the edge's endpoint coordinates according to the decrease in contour level along the edge. The coordinates and drawing instructions generated for the contouring trees of Figures 5a and 7a are represented in Figures 5b and 7b.

There are some subtleties not evident from the above that are best detailed using a pseudocode description of the traversal algorithm. Figure 8 depicts the traversal procedure for the contouring tree assuming a particular data organization. The notation is quite standard. The pointers to the descendent nodes of NODE are LEFT(NODE), MIDDLE(NODE), and RIGHT(NODE). For each node of the contouring tree, there are three pieces of information: the value associated with the node, VALUE(NODE), the coordinate associated with the node, XYZ(NODE), and the connectivity associated with the node, CONN(NODE).

The generation of coordinates and drawing instructions from a contouring tree begins with routine CONTOUR_SUBGRID of Figure 8. That routine receives a pointer to the root node of the contouring tree. It then starts the traversal by calling routine VISIT with that root node. Routine VISIT checks to see if the edge defined by the passed in node and that node's ancestor, NODE and ANCESTOR, contains the contour level. If the edge does contain the contour level, the edge intersection coordinate is computed using linear interpolation and issued to the display along with the connectivity associated with that node, CONN(NODE). If we issue a coordinate and connectivity for a node, we need to check the subtree under that node for equivalued edges. If an equivalued edge at the contour level is found, a coordinate and drawing instruction pair are issued for that equivalued edge (routine VISIT_SUBTREE). Once a coordinate and drawing instruction pair have been issued for an edge, and once the subtree beneath that edge has been investigated for equivalued edges, further traversal of that subtree is terminated. If an edge is found not to contain the contour level, the traversal continues as depicted at the bottom of routine VISIT.

The pre-order traversal procedure described above generates the coordinates and drawing instructions for the part of the 2 x 2 subgrid the contouring tree represents. To generate the coordinates for a larger two-dimensional grid, we generate the contouring trees for each 2 x 2 subgrid of that grid, and then apply the traversal procedure to those trees. We note here that no ordering is required in the generation of coordinates for the 2 x 2 subgrids. The coordinate and drawing instruction set generated for each 2 x 2 subgrid is complete and independent of the picture generated for any neighboring 2 x 2.

Contouring Tree Description

Pointers to descendent nodes:

LEFT(NODE)
MIDDLE(NODE)
RIGHT(NODE)

Values associated with each node:

VALUE(NODE): grid value
XYZ(NODE) : coordinate of that grid value.
CONN(NODE) : drawing instruction.

```
Procedure CONTOUR_SUBGRID(ROOT)

  VISIT(ROOT,ROOT)  # begin the traversal of the pointed at
                    # contouring tree.

end.



Procedure VISIT(NODE,ANCESTOR)

  if(NODE == NULL)
  {
    return
  }

  if((VALUE(NODE) <= CONTOUR_LEVEL < VALUE(ANCESTOR))
     OR
    (VALUE(NODE) == CONTOUR_LEVEL AND NODE == ANCESTOR))
  {

    # Edge contains the contour level.

    Issue a coordinate computed via linear interpolation
    along the edge.

    Issue CONN(NODE) as the drawing instruction.
```

Figure 8
Pseudocode of the Traversal Algorithm for the Contouring Tree

```
    # Check subtrees of this node for equivalued edges.
    VISIT_SUBTREE(LEFT(NODE),NODE)
    VISIT_SUBTREE(MIDDLE(NODE),NODE)
    VISIT_SUBTREE(RIGHT(NODE),NODE)

    return   # no need to examine the subtree further.

  }  # endif coordinates were generated for an edge.


  VISIT(LEFT(NODE),NODE)   # visit left subtree.
  VISIT(MIDDLE(NODE),NODE)   # visit middle subtree.
  VISIT(RIGHT(NODE),NODE)   # visit right subtree.

  return

end



Procedure VISIT_SUBTREE(SUBNODE,SUBANCESTOR)

  if(SUBNODE == NULL)
  {
    return
  }

  if(VALUE(SUBNODE) == CONTOUR_LEVEL)
  {

    Issue coordinates for the equivalued edge.
    Setpoint on XYZ(SUBANCESTOR).
    Drawto     XYZ(SUBNODE).

  }


  VISIT_SUBTREE(LEFT(SUBNODE),SUBNODE)
  VISIT_SUBTREE(MIDDLE(SUBNODE),SUBNODE)
  VISIT_SUBTREE(RIGHT(SUBNODE),SUBNODE)

  return

end
```

Figure 8 (continued)
Pseudocode of the Traversal Algorithm for the Contouring Tree

4.1. Contouring Tree Use Discussion

Having presented the use of the contouring tree, we must look back and discuss its capabilities and limitations. The initial impression is that the contouring tree provides a nice, uniform framework for generating the coordinates and drawing instructions appropriate to the 2 x 2 subgrid. The algorithm takes care of the single contour crossing cases quite readily. The algorithm also takes care of the difficult two contours crossing case for the 2 x 2 subgrid. The algorithm correctly handles subgrids containing equivalued lines at the contour level. The algorithm also handles subgrids containing a single grid point at the contour level.

The core problems with this algorithm all concern issues of picture efficiency. Since the display generated for each 2 x 2 subgrid is generated independently of any neighboring 2 x 2 subgrids, equivalued lines at the contour level on the border of a subgrid will be duplicated. A similar problem occurs for subgrid corner values that equal the contour level. If we display either of the above cases on a calligraphic display device, we will see a bright line for the equivalued edge, and a bright point for the grid value equal to the contour level. Another problem, also due to the independent computation of each 2 x 2 subgrid, is that no ordering is provided for coordinates that come out of this algorithm. For calligraphic displays, this is a problem because for such devices electron beam movement is expensive. A contour display that causes the maximum movement of the electron beam every other subgrid greatly decreases the the vector capability of the calligraphic display device.

There are three possible solutions to the first problem, that of duplicate vectors. The easiest solution is to choose an output display device for which such picture inefficiencies do not matter, i.e. a raster display. Vector ordering is also eliminated as a problem with this solution. The second solution to the vector duplication problem is to set aside points and lines at the contour level that correspond to subgrid boundaries. A final pass at the end of the computation for a complete two-dimensional plane could readily cull the duplicates. This second solution does nothing for the vector ordering problem. If this solution to the vector duplicates problem is utilized, one either doesn't worry about the vector ordering, or performs a sort on the vectors. The third solution, and the most expensive of the three, is to merge the set of trees generated for the two-dimensional grid such that duplicate edges in separate trees are eliminated. This solution has the added benefit that the resultant contours are generated in an order that solves the beam movement problem. This solution is not described in detail here and the reader is referred to [Zyda,1981] for further detail. For this study, the first and simplest solution is assumed, and consequently, the expected output display device is the raster display.

5. Contouring Tree Construction

Contouring tree construction is best understood if we describe that procedure in graph theoretic terms. We begin by assuming that we have a graph of five nodes, each node being one of the subgrid definition nodes or the center node of average value. The eight edges on that graph are the subgrid boundary edges, and the edges from each subgrid definition point to the center point of average value. We can readily assign directions to each edge of this graph using the values assigned to each node. Equivalued edges can be assigned an arbitrary direction. (One such assignment is to make equivalued edges along the border point in a counterclockwise fashion, and equivalued edges from the center point in towards the center.) With these assumptions, each 2 x 2 subgrid is perceived as a directed graph. The question then becomes, how do we obtain

```
    # Check subtrees of this node for equivalued edges.
    VISIT_SUBTREE(LEFT(NODE),NODE)
    VISIT_SUBTREE(MIDDLE(NODE),NODE)
    VISIT_SUBTREE(RIGHT(NODE),NODE)

    return   # no need to examine the subtree further.

  }  # endif coordinates were generated for an edge.


  VISIT(LEFT(NODE),NODE)   # visit left subtree.
  VISIT(MIDDLE(NODE),NODE)   # visit middle subtree.
  VISIT(RIGHT(NODE),NODE)   # visit right subtree.

  return

end



Procedure VISIT_SUBTREE(SUBNODE,SUBANCESTOR)

  if(SUBNODE == NULL)
  {
    return
  }

  if(VALUE(SUBNODE) == CONTOUR_LEVEL)
  {

    Issue coordinates for the equivalued edge.
    Setpoint on XYZ(SUBANCESTOR).
    Drawto     XYZ(SUBNODE).

  }


  VISIT_SUBTREE(LEFT(SUBNODE),SUBNODE)
  VISIT_SUBTREE(MIDDLE(SUBNODE),SUBNODE)
  VISIT_SUBTREE(RIGHT(SUBNODE),SUBNODE)

  return

end
```

Figure 8 (continued)
Pseudocode of the Traversal Algorithm for the Contouring Tree

4.1. Contouring Tree Use Discussion

Having presented the use of the contouring tree, we must look back and discuss its capabilities and limitations. The initial impression is that the contouring tree provides a nice, uniform framework for generating the coordinates and drawing instructions appropriate to the 2 x 2 subgrid. The algorithm takes care of the single contour crossing cases quite readily. The algorithm also takes care of the difficult two contours crossing case for the 2 x 2 subgrid. The algorithm correctly handles subgrids containing equivalued lines at the contour level. The algorithm also handles subgrids containing a single grid point at the contour level.

The core problems with this algorithm all concern issues of picture efficiency. Since the display generated for each 2 x 2 subgrid is generated independently of any neighboring 2 x 2 subgrids, equivalued lines at the contour level on the border of a subgrid will be duplicated. A similar problem occurs for subgrid corner values that equal the contour level. If we display either of the above cases on a calligraphic display device, we will see a bright line for the equivalued edge, and a bright point for the grid value equal to the contour level. Another problem, also due to the independent computation of each 2 x 2 subgrid, is that no ordering is provided for coordinates that come out of this algorithm. For calligraphic displays, this is a problem because for such devices electron beam movement is expensive. A contour display that causes the maximum movement of the electron beam every other subgrid greatly decreases the the vector capability of the calligraphic display device.

There are three possible solutions to the first problem, that of duplicate vectors. The easiest solution is to choose an output display device for which such picture inefficiencies do not matter, i.e. a raster display. Vector ordering is also eliminated as a problem with this solution. The second solution to the vector duplication problem is to set aside points and lines at the contour level that correspond to subgrid boundaries. A final pass at the end of the computation for a complete two-dimensional plane could readily cull the duplicates. This second solution does nothing for the vector ordering problem. If this solution to the vector duplicates problem is utilized, one either doesn't worry about the vector ordering, or performs a sort on the vectors. The third solution, and the most expensive of the three, is to merge the set of trees generated for the two-dimensional grid such that duplicate edges in separate trees are eliminated. This solution has the added benefit that the resultant contours are generated in an order that solves the beam movement problem. This solution is not described in detail here and the reader is referred to [Zyda,1981] for further detail. For this study, the first and simplest solution is assumed, and consequently, the expected output display device is the raster display.

5. Contouring Tree Construction

Contouring tree construction is best understood if we describe that procedure in graph theoretic terms. We begin by assuming that we have a graph of five nodes, each node being one of the subgrid definition nodes or the center node of average value. The eight edges on that graph are the subgrid boundary edges, and the edges from each subgrid definition point to the center point of average value. We can readily assign directions to each edge of this graph using the values assigned to each node. Equivalued edges can be assigned an arbitrary direction. (One such assignment is to make equivalued edges along the border point in a counterclockwise fashion, and equivalued edges from the center point in towards the center.) With these assumptions, each 2 x 2 subgrid is perceived as a directed graph. The question then becomes, how do we obtain

the contouring tree, or trees from this directed graph? We can put this question in terms of graph theory if we notice that a contouring tree is a directed tree. The problem then becomes one of obtaining the directed tree, or trees, from the directed graph such that the order of edge attachment in the tree corresponds to the order in the directed graph (the 2 x 2 subgrid). From graph theory, we have the requirement that a directed tree has the in-degree of its root node equal to zero, and the in-degree of every other node equal to one [Even,1979]. To examine the in-degree of each node of the directed graph, we must construct the in-degree matrix D for that graph. The in-degree matrix D of a directed graph G is defined in [Even,1979] as:

D(i,j) = in-degree(i), if i = j.

D(i,j) = -k, if i is not equal to j,
where k is the number of
edges in G from i to j
(i.e., -1 for all our graphs).

Figures 9 and 10 show the in-degree matricies for our example 2 x 2 subgrids.

From the figures, we note that the roots of the contouring trees are recognizable from D as D(i,i) = 0. This matches the first part of the directed tree requirement. Further examination of the diagonal of the in-degree matricies introduces two difficulties in our attempts to convert the directed graphs into directed trees: (1) multiple roots (in-degree(v) = 0 for more than one node) and (2) in-degree(v) > 1 for some nodes v. (Note: we have assumed that we have the structure represented by the directed graph and that we can manipulate it.)

The first problem, that of multiple roots, is handled by producing multiple sets of verticies and multiple in-degree matricies such that there is only one root per in-degree matrix. For our case, the maximum number of roots for a single 2 x 2 subgrid is two. We eliminate this problem by alternately removing each root node from the complete set of verticies and all edges attached to that deleted node and then making separate in-degree matricies. The second problem in the conversion of the directed graph into a directed tree, that of in-degree(v) > 1 for some nodes, is resolved by node duplication. For each diagonal entry D(i,i) = n, where n > 1, we create n-1 duplicates of that node, for a total of n, taking care to copy the appropriate values, coordinates, etc. We then reassign the original edges that went to the single node, such that each edge receives its own copy of the duplicated node. The edges that are reassigned are those between each node of column i of D that has a -1 and each of the n duplicate nodes. When performed for each in-degree matrix created, this operation forms a new directed graph that is the desired directed tree

5.1. Drawing Command Placement

The above section detailed the creation of the contouring trees for the 2 x 2 subgrid utilizing nothing more than basic graph definitions, and simple construction procedures. The only thing remaining for completing this construction is the placement of the drawing commands into the contouring trees.

Drawing commands are placed in the contouring tree to indicate when a line enters the region represented by the contouring tree either from a neighboring subgrid or from a location off of the grid. If we look at the structure of the contouring tree, such as that exhibited in Figure 5a, and consider that during the traversal, the edges are examined in a counterclockwise, and downward

D(i,j)	1	2	3	4	5
1	0	-1	0	-1	-1
2	0	3	0	0	0
3	0	-1	1	-1	0
4	0	0	0	3	0
5	0	-1	-1	-1	1

Figure 9
In-degree Matrix for the Directed Graph Superimposed on a 2 x 2 Subgrid

D(i,j)	1	2	3	4	5
1	3	0	0	0	0
2	-1	0	-1	0	-1
3	0	0	3	0	0
4	-1	0	-1	0	-1
5	-1	0	-1	0	2

D(i,j)	1	3	4	5	D(i,j)	1	2	3	5
150					90				
1	2	0	0	0	1	2	0	0	0
3	0	2	0	0	2	-1	0	-1	-1
4	-1	-1	0	-1	3	0	0	2	0
5	-1	-1	0	1	5	-1	0	-1	1

Figure 10
In-degree Matrix for the Directed Graph Superimposed
on a 2 x 2 Subgrid with Saddle Point

ordering from the root, we note that we need to place setpoint drawing commands on the lower valued node of each edge that presents a new lowest value for the tree. (Note that the drawing command setpoint indicates to the display device that it should move its "drawing instrument", i.e. electron beam, pen, etc., in a non-drawing mode to the specified location, and that it should then place that drawing instrument into a drawing mode. Drawto indicates to the display device that it should move its drawing instrument in a drawing mode to the specified location. Drawpoint indicates to the display device that it should move its drawing instrument in a non-drawing mode to the specified location, and that it should then turn that drawing instrument on for the space of a single point.) We insert these drawing commands by way of a pre-order traversal of the directed tree, placing a setpoint command on each node that is a new lowest value for the tree. This drawing command placement strategy is based upon the fact that if we have a contour level for which we desire a picture, the first drawing command we generate for any contouring tree is a setpoint. Although fairly effective, this procedure does not provide a complete solution to drawing command insertion. Some neighboring edges in the contouring tree, i.e. edges sharing an ancestor node, have a "split" between them, i.e., the edges are not immediate counterclockwise neighbors in the original grid. In this case, we must indicate the discontinuity in the contouring tree. We register the discontinuity on the lower valued node of the edge where the discontinuity occurs. For example, in Figure 5a the edges (3,3)-(3,2) and (3,3)-(2,3) are neighbors in the contouring tree but are not immediate neighbors in the original grid. We indicate this split by placing a "1" on the lower valued node of edge (3,3)-(2,3).

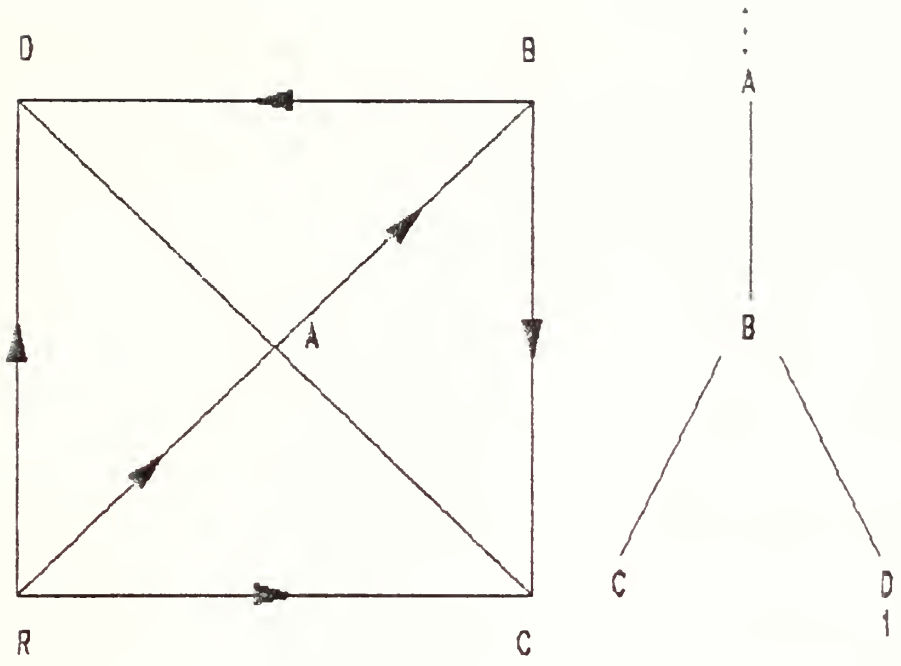
In order to recognize the nodes that require a drawing command indicating a split edge in the contouring tree, we must first examine where split edges occur in the 2 x 2 subgrid. These occur in the 2 x 2 subgrid wherever the subgrid has edge directions and tree edge configurations as depicted in Figure 11, i.e. where there are neighboring tree edges not directly corresponding to neighboring subgrid edges. There are two cases in the figure.

In case 1, the split edge is edge (B,D). This edge neighbors edge (B,C) in the contouring tree but is not the immediate counterclockwise neighbor edge of (B,C). The lower valued node of edge (B,D), D, receives a setpoint drawing command. From the figure, we see that node D has a possibility of being a "sink", i.e., a node with all incoming edges. This depends upon the the direction of edge (A,D). There are two cases to consider: (1a) A —> D or (1b) D —> A. Case 1a, A —> D, is easy to show as possible because it may be seen in Figure 5a. Case 1b, D —> A, is somewhat more difficult, because we must show that it cannot possibly occur within the relations specified on Figure 11. We begin by compiling a small set of the given relations on the directed graph shown in the figure:

$$R \geq C$$
$$R \geq A$$
$$R \geq D$$
$$B \geq D$$
$$A \geq D$$

Case 1b then becomes the question, is it possible to get $D \geq A$? Assume that $D \geq A$ is possible. Now we have $B \geq D$ so we can replace D with B. We get $B \geq A$, which is a contradiction of our original given, $A \geq B$, unless $A = B$. So we cannot have D —> A unless A = B. Can we eliminate the possibility of the edge from A to B pointing from the center outward in the case where A = B? We can do this when we set up the original directed graph, biasing edge selection of constant

CASE 1



CASE 2

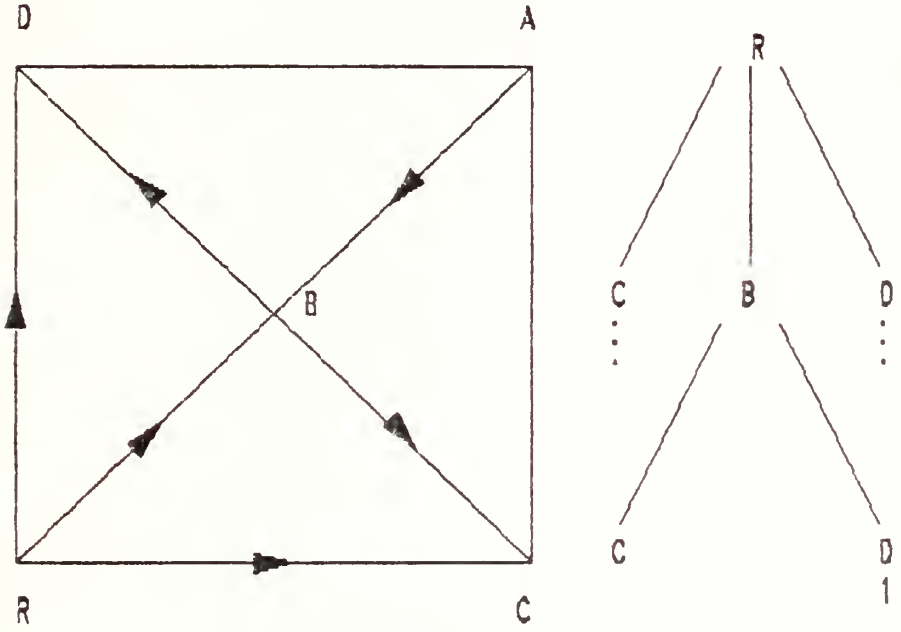


FIGURE 11
ALL SPLIT EDGE POSSIBILITIES FOR THE 2 X 2 SUBGRID

valued edges to point always towards the center. Since the direction assigned to a constant edge is, in fact, arbitrary, we can assume that we never see the condition D ——> A, given the set of fixed directions assigned to case 1. Hence, for case 1 of Figure 11, a split edge node is indicated during pre-order traversal of the directed tree whenever we encounter, for the first time, a node representing a sink grid point. We now show a similar finding for case 2.

In case 2 of Figure 11, the split edge is edge (B,D). This edge neighbors edge (B,C) in the contouring tree but is not the immediate counterclockwise neighbor edge of (B,C). The lower valued node of edge (B,D), D, receives a set-point drawing command. From the figure, we see that node D is possibly a sink. This depends on the direction of edge (A,D). There are two cases to consider: (2a) A ——> D or (2b) D ——> A. Case 2a, A ——> D, occurs in Figure 7a for the two maxima example so we know this case is possible. Case 2b, D ——> A, is more difficult because we must show that it cannot occur. We list a few relations describing the partial directed graph of the figure:

$$R \geq C$$
$$R \geq B$$
$$R \geq D$$
$$B \geq C$$
$$A \geq B$$
$$B \geq D$$

The question that arises is whether it is possible to get $D \geq A$? Assume $D \geq A$ is possible. Now we have $B \geq D$ so we can replace D with B. This gives $B \geq A$, which is a contradiction of our original given, $A \geq B$, unless $A = B$. So we cannot have D ——> A unless A = B. Clearly we already have eliminated the possibility of the edge pointing from B to A by our initial configuration and by our biased edge selection for constant edges. Hence, for case 2 of Figure 11, a split edge node is indicated during pre-order traversal of the directed tree whenever we encounter, for the first time, a node representing a sink grid point.

There are no other split edge configurations possible in the 2 x 2 subgrid. If we add a procedure that checks not only the new lowest value in the directed tree but also the first encounter with a sink grid node during the same pre-order traversal, we will correctly place the drawing commands in the directed tree, converting it into the desired contouring tree.

6. Completeness for the Subgrid Contouring Algorithm

An algorithm is complete with respect to the subgrid contour generation problem if it always generates the expected picture for any proffered subgrid. Since it is impossible to test a subgrid contouring algorithm by trying it against all possible subgrids, we must rely on another mechanism. Our ability to characterize the total number of possible contour crossing cases in as small a number as ten brings to mind such a mechanism for validating the contouring tree algorithm.

For each 2 x 2 case possibility, show that all possible contouring trees generatable from the relations specified for the case either

> (1) produce the equivalent picture (to the expected),
>
> (2) or are in actuality the contouring trees of another case, i.e. we show that the equivalent set of coordinates is impossible to generate from the contouring tree.

For each subgrid crossing case...
{

> For each boundary point of the 2 x 2 subgrid...
> {
>
> > Use the boundary point as the root of the tree.
> >
> > Generate the 32 possible trees that have that root (and initial tree configuration).
> >
> > For each tree...
> > {
> >
> > > Can we obtain a set of coordinates and drawing instructions equivalent to the expected set for this 2 x 2 case?
> > >
> > > If we have the equivalent picture, then we are OK. Next.
> > >
> > > If we can't get the equivalent picture, we must show that we violate the relations set up for the problem. This implies a different case. Next. If we don't violate the relations, that implies a problem with the contouring tree algorithm.
> >
> > }
> }
}

Figure 12
Outline of the Contouring Tree Algorithm Completeness Proof

If we are able to show for each of the ten cases of Figure 4 that all possible contouring trees generatable from the relations specified for each case either

> (1) produce a picture equivalent to that expected for the case,
>
> (2) or are in actuality the contouring trees of a different case, i.e. we show that the expected set of coordinates and drawing instructions is impossible to generate from the contouring tree (we violate the given relations),

then we will have shown completeness for the algorithm. This is a fairly large proof, an outline of which can be seen in Figure 12.

The idea behind the proof for an individual case is to use each boundary point of the subgrid as the root of the set of all possible trees generatable from that root. When we fix the root for a contouring tree, we fix the three immediate descendents of that root and the three edges to those descendents. This leaves unspecified the placement of five edges in the contouring tree. If we assume that each edge direction is possible, this means that once we fix the root of a tree, there are 32 possible trees that can be generated. If we cycle through each boundary point as the root of the tree, there are 32 times 4 possible trees for each case. For each of the 128 possible trees, we then check to see if we can obtain a set of coordinates and drawing instructions equivalent to the expected set for the case (see Figure 4). If we obtain the equivalent picture from a tree, we go on to the next tree (or case). If we cannot obtain the equivalent picture, we must show that we violate the relations given for the case. Such a violation implies that we have a different case, and hence, can go on to the next tree (or case). The negative indications for the proof are if we cannot get the equivalent picture but do not violate the relations set up for the case. Such an indication would imply a problem with the contouring tree algorithm.

The magnitude of this proof is immense. There are ten different subgrid cases, each with 128 possible contouring trees. The evaluation of this total set of trees has been performed utilizing a program that systematically traverses each possible tree, checking for either the equivalent picture, or a violation of the given relations. The results of this study indicate that the contouring algorithm is complete.

7. Conclusions

This study has described an algorithm for generating the contour lines and drawing instructions for a single 2 x 2 subgrid in a manner that is independent from the calculations required for any other 2 x 2 subgrid. This algorithm has been shown to be part of a larger algorithm used to generate the contours for two-dimensional grids composed of multiple 2 x 2 subgrids. It has also been shown to be part of an even larger algorithm used to generate the contour surface display for three-dimensional grids composed of multiple, parallel two-dimensional grids. The fact that we can generate the contour lines for each 2 x 2 subgrid independently from the calculations required for any other 2 x 2 subgrid means that our algorithm for the generation of contour surface displays is highly decomposable. This becomes quite striking if we consider that a typical 30 x 30 x 30 three-dimensional grid has 75,690 2 x 2 subgrids. For such a case, there is a potential for 75,690 concurrent computations. Given a technology that allows such parallelism and assuming that we can both load and unload the data from that system, it appears as if this algorithm has the potential for

speeding up one of the most frequently used graphics computations.

8. References

Barry, C.D. and Sucher, J. H. "Interactive Real-Time Contouring of Density Maps," American Crystallographic Association Winter Meeting, Honolulu, March 1979, Poster Session.

Cottafava, G. and Le Moli, G. "Automatic Contour Map," *Communications of the ACM*, Vol. 12, No. 7 (July 1969), pp. 386-391.

Dayhoff, M.O. "A Contour-Map Program for X-Ray Crystallography," *Communications of the ACM*, Vol. 6, No. 10 (October 1963), pp.620-622.

Dutton, G., "An Extensible Approach to Imagery of Gridded Data," *Computer Graphics: A Quarterly Report of SIGGRAPH-ACM*, Vol. 11, No. 2 (July 1977), p. 159.

Even, Shimon *Graph Algorithms*, Potomac, Maryland: Computer Science Press, 1979.

Faber, D.H., Rutten-Keulemans, E.W.M., and Altona, C. "Computer Plotting of Contour Maps: An Improved Method," *Computers & Chemistry*, Vol. 3, pp. 51-55, Great Britain: Pergamon Press Ltd., 1979.

Gold, G.M., "Automated Contour Mapping Using Triangular Element Data Structures and An Interpolant Over Each Irregular Triangular Domain," *Computer Graphics: A Quarterly Report of SIGGRAPH-ACM*, Vol. 11, No. 2 (July 1977), p. 170.

McLain, D.H., "Drawing Contours from Arbitrary Data Points," *The Computer Journal*, Vol.17, No. 4 (1974), p. 318.

Sabin, M.A., "Contouring -- A Review of Methods for Scattered Data," *Mathematical Methods in Computer Graphics and Design*, Edited by K.W. Brodlie, pp. 63-86, Great Britain: Academic Press, 1980.

Sutcliffe, D.C., "Contouring Over Rectangular and Skewed Rectangular Grids – An Introduction," *Mathematical Methods in Computer Graphics and Design*, Edited by K.W. Brodlie, pp. 39-62, Great Britain: Academic Press, 1980.

Sutcliffe, D.C., "A Remark on a Contouring Algorithm," *The Computer Journal*, Vol. 19, No. 4 (1976a), p. 333.

Sutcliffe, D.C., "An Algorithm for Drawing the Curve f(x,y) = 0," *The Computer Journal*, Vol. 19, No. 3, (1976b), p. 246.

Wright, Thomas and Humbrecht, John "ISOSRF – An Algorithm for Plotting Iso-Valued Surfaces of a Function of Three Variables," *Computer Graphics: A Quarterly Report of SIGGRAPH-ACM*, Vol. 13, No. 2 (August 1979), pp. 182-189.

Zyda, Michael J. *Algorithm Directed Architectures for Real-Time Surface Display Generation*, D.Sc. Dissertation, Dept. of Computer Science, Washington Univ, St. Louis, Missouri, 1984.

Zyda, Michael J. "A Contour Display Generation Algorithm for VLSI Implementation," *Selected Reprints on VLSI Technologies and Computer Graphics*, Compiled by Henry Fuchs, p. 459, Silver Spring, Maryland: IEEE Computer Society Press, 1983.

Zyda, Michael J. "A Contour Display Generation Algorithm for VLSI Implementation," *Computer Graphics: A Quarterly Report of SIGGRAPH-ACM*, Vol. 16, No. 3 (July 1982), p. 135.

Zyda, Michael J. "Multiprocessor Considerations in the Design of a Real-Time Contour Display Generator," Technical Memorandum 42, St. Louis: Department of Computer Science, Washington University, December 1981.

INITIAL DISTRIBUTION LIST

Defense Technical Information Center Cameron Station Alexandria, VA 22314	2
Dudley Knox Library Code 0142 Naval Postgraduate School Monterey, CA 93943	3
Office of Research Administration Code 012A Naval Postgraduate School Monterey, CA 93943	1
Chairman, Code 52Hq Department of Computer Science Naval Postgraduate School Monterey, CA 93943	40
Assistant Professor Michael J. Zyda, Code 52Zy Department of Computer Science Naval Postgraduate School Monterey, CA 93943	40